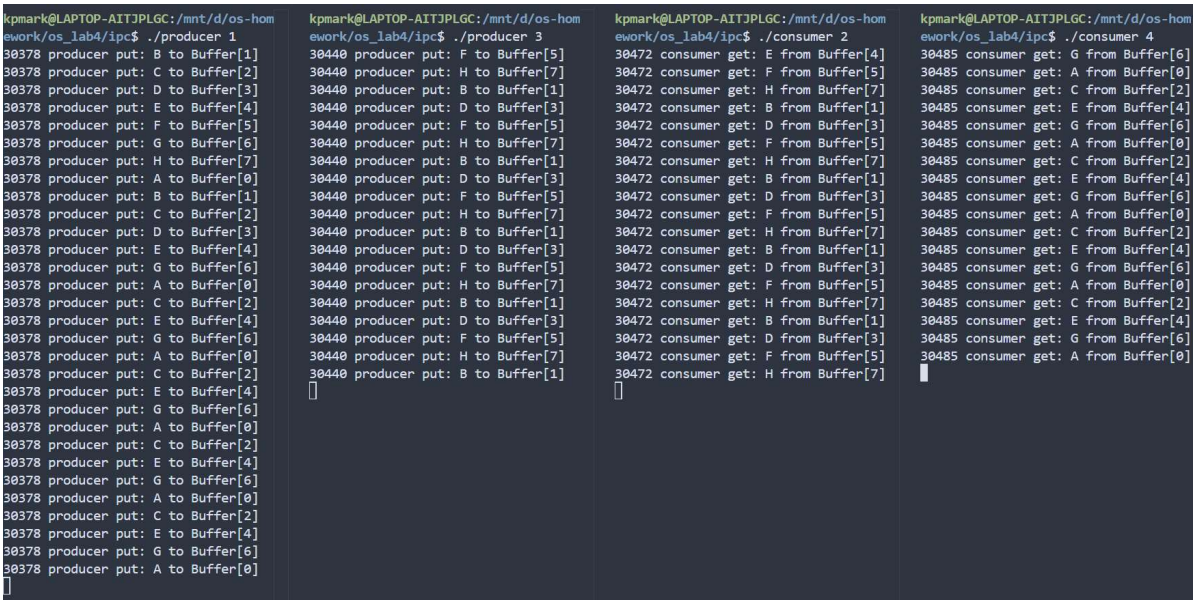
第三次上机报告(lab_04)

进程同步机制的实现原理

示例代码执行结果如下图



System V IPC机制分析

从实验代码中可以看出，Linux系统使用System V IPC（Inter-Process Communication）机制提供了三种主要的进程间通信方式：`

信号量（Semaphore）

```
// 创建信号量
int set_sem(key_t sem_key, int sem_val, int sem_flg) {
    int sem_id;
    Sem_uns sem_arg;

    if ((sem_id = semget(sem_key, 1, sem_flg)) < 0) {
        perror("semaphore create error");
        exit(EXIT_FAILURE);
    }

    sem_arg.val = sem_val;
    if (semctl(sem_id, 0, SETVAL, sem_arg) < 0) {
        perror("semaphore set error");
        exit(EXIT_FAILURE);
    }
    return sem_id;
}
```

共享内存（Shared Memory）

```
// 创建共享内存
char* set_shm(key_t shm_key, int shm_num, int shm_flg) {
    int shm_id;
    char* shm_buf;

    if ((shm_id = shmget(shm_key, shm_num, shm_flg)) < 0) {
        perror("shareMemory set error");
        exit(EXIT_FAILURE);
    }

    if ((shm_buf = (char*)shmat(shm_id, 0, 0)) < (char*)0) {
        perror("get shareMemory error");
        exit(EXIT_FAILURE);
    }
    return shm_buf;
}
```

消息队列（Message Queue）

虽然在当前实验中未使用，但代码框架中包含了消息队列的支持。

生产者/消费者问题的同步机制分析

问题模型

生产者/消费者问题涉及：

- **共享资源**：有界缓冲区（循环队列）
- **进程类型**：生产者进程、消费者进程
- **同步要求**：
    - 生产者不能向满缓冲区放入产品
    - 消费者不能从空缓冲区取出产品
    - 同时只能有一个进程访问缓冲区

信号量设计

从实验代码中可以看出，使用了四个信号量：

```
// 生产者有关的信号量
int prod_sem;  // 生产者同步信号量，初值为缓冲区大小
int pmtx_sem;  // 生产者互斥信号量，初值为1

// 消费者有关的信号量
int cons_sem;  // 消费者同步信号量，初值为0
int cmtx_sem;  // 消费者互斥信号量，初值为1
```

生产者算法分析

```
while (1) {
    // 如果缓冲区满则生产者阻塞
    down(prod_sem);            // P(empty) - 等待空位
    // 如果另一生产者正在放产品，本生产者阻塞
    down(pmtx_sem);            // P(mutex) - 进入临界区

    // 临界区：放入产品
    buff_ptr[*pput_ptr] = 'A' + *pput_ptr;
    printf("%d producer put: %c to Buffer[%d]\n",
           getpid(), buff_ptr[*pput_ptr], *pput_ptr);
    *pput_ptr = (*pput_ptr + 1) % buff_num;

    // 唤醒阻塞的生产者
    up(pmtx_sem);              // V(mutex) - 离开临界区
    // 唤醒阻塞的消费者
    up(cons_sem);              // V(full) - 增加产品数
}
```

消费者算法分析

```
while (1) {
    // 如果无产品消费者阻塞
    down(cons_sem);            // P(full) - 等待产品
    // 如果另一消费者正在取产品，本消费者阻塞
    down(cmtx_sem);            // P(mutex) - 进入临界区

    // 临界区：取出产品
    printf("%d consumer get: %c from Buffer[%d]\n",
           getpid(), buff_ptr[*cget_ptr], *cget_ptr);
    *cget_ptr = (*cget_ptr + 1) % buff_num;

    // 唤醒阻塞的消费者
    up(cmtx_sem);              // V(mutex) - 离开临界区
    // 唤醒阻塞的生产者
    up(prod_sem);              // V(empty) - 增加空位数
}
```

抽烟者问题的同步机制分析

代码执行结果如下图

```
kpmark@LAPTOP-AITJPLGC:/mnt/d/os-homework/os_lab4/smoker$ make run
启动抽烟者问题演示...
./main
供应者 1 启动
有胶水的抽烟者 启动 (拥有: 胶水)
有胶水的抽烟者: 等待材料...
供应者 1: 提供了 纸 和 胶水 (缺 烟草)
有纸的抽烟者 启动 (拥有: 纸)
有纸的抽烟者: 等待材料...
有烟草的抽烟者 启动 (拥有: 烟草)
供应者 2 启动
供应者 2: 提供了 纸 和 胶水 (缺 烟草)
有烟草的抽烟者: 等待材料...
有烟草的抽烟者: 发现有材料，开始卷烟...
有烟草的抽烟者: 正在卷烟... (第 1 支烟)
有烟草的抽烟者: 正在抽烟...
有烟草的抽烟者: 抽完了！通知供应者继续工作
供应者 1: 提供了 烟草 和 胶水 (缺 纸)
有纸的抽烟者: 发现有材料，开始卷烟...
有纸的抽烟者: 正在卷烟... (第 2 支烟)
有烟草的抽烟者: 等待材料...
有纸的抽烟者: 正在抽烟...
有纸的抽烟者: 抽完了！通知供应者继续工作
供应者 2: 提供了 烟草 和 纸 (缺 胶水)
有胶水的抽烟者: 发现有材料，开始卷烟...
有胶水的抽烟者: 正在卷烟... (第 3 支烟)
有纸的抽烟者: 等待材料...
有胶水的抽烟者: 正在抽烟...
有胶水的抽烟者: 抽完了！通知供应者继续工作
供应者 1: 提供了 纸 和 胶水 (缺 烟草)
有烟草的抽烟者: 发现有材料，开始卷烟...
有烟草的抽烟者: 正在卷烟... (第 4 支烟)
```

问题模型

抽烟者问题包含：

- **参与者**：3个抽烟者（分别拥有烟草、纸、胶水之一）、2个供应者
- **共享资源**：桌子（存放材料）
- **同步要求**：
    - 供应者每次提供两种不同的材料
    - 拥有第三种材料的抽烟者可以取材料制烟
    - 抽烟完成后通知供应者继续工作

信号量设计

```
// 为每种抽烟者设置同步信号量
int tobacco_sem;  // 有烟草的抽烟者信号量，初值为0
int paper_sem;    // 有纸的抽烟者信号量，初值为0
int glue_sem;     // 有胶水的抽烟者信号量，初值为0
int supplier_sem; // 供应者信号量，初值为1
int mutex_sem;    // 互斥信号量，初值为1
```

供应者算法分析

```
while (status->active) {
    sem_wait(supplier_sem);   // 等待允许工作
    sem_wait(mutex_sem);      // 进入临界区

    // 清空桌子并提供材料
    memset(table->materials, 0, sizeof(table->materials));
    int combination = table->round % 3;

    switch (combination) {
        case 0:  // 提供纸和胶水
            table->materials[PAPER] = 1;
            table->materials[GLUE] = 1;
            sem_signal(tobacco_sem);  // 唤醒有烟草的抽烟者
            break;
        case 1:  // 提供烟草和胶水
            table->materials[TOBACCO] = 1;
            table->materials[GLUE] = 1;
            sem_signal(paper_sem);    // 唤醒有纸的抽烟者
            break;
        case 2:  // 提供烟草和纸
            table->materials[TOBACCO] = 1;
            table->materials[PAPER] = 1;
            sem_signal(glue_sem);     // 唤醒有胶水的抽烟者
            break;
    }

    table->round++;
    sem_signal(mutex_sem);        // 离开临界区
}
```

抽烟者算法分析

```
while (status->active) {
    sem_wait(my_sem);             // 等待所需材料

    if (!status->active) break;

    sem_wait(mutex_sem);          // 进入临界区

    // 取走材料并更新状态
    memset(table->materials, 0, sizeof(table->materials));
    status->total_smokes++;
    int current_smokes = status->total_smokes;

    sem_signal(mutex_sem);        // 离开临界区
```

```
    // 制烟和抽烟过程
    printf("%s: 正在卷烟... (第 %d 支烟)\n", my_name, current_smokes);
    sleep(2);
    printf("%s: 正在抽烟... \n", my_name);
    sleep(2);

    // 检查是否达到终止条件
    if (current_smokes >= 10) {
        status->active = 0;
        // 唤醒所有等待的进程
        sem_signal(tobacco_sem);
        sem_signal(paper_sem);
        sem_signal(glue_sem);
        sem_signal(supplier_sem);
        break;
    }

    sem_signal(supplier_sem);      // 通知供应者继续工作
}
```

信号量机制的工作原理

信号量的定义和操作

信号量是一个整数变量，配合两个原子操作：

- **P操作（down/wait）**：sem = sem - 1，如果sem < 0则进程阻塞
- **V操作（up/signal）**：sem = sem + 1，如果sem ≤ 0则唤醒一个等待进程

P操作的实现

```
int down(int sem_id) {  // P操作
    struct sembuf buf;
    buf.sem_op = -1;       // 信号量减1
    buf.sem_num = 0;       // 操作第0个信号量
    buf.sem_flg = SEM_UNDO; // 进程终止时自动撤销操作

    if ((semop(sem_id, &buf, 1)) < 0) {
        perror("down error ");
        exit(EXIT_FAILURE);
    }
    return EXIT_SUCCESS;
}
```

V操作的实现

```c
int up(int sem_id) {  // V操作
    struct sembuf buf;
    buf.sem_op = 1;         // 信号量加1
    buf.sem_num = 0;        // 操作第0个信号量
    buf.sem_flg = SEM_UNDO; // 进程终止时自动撤销操作

    if ((semop(sem_id, &buf, 1)) < 0) {
        perror("up error ");
        exit(EXIT_FAILURE);
    }
    return EXIT_SUCCESS;
}
```

信号量初值和变化的物理意义

生产者/消费者问题中的信号量

信号量	初值	物理意义	值的变化
prod_sem	buff_num (8)	缓冲区中空闲位置数	生产时减1，消费时加1
cons_sem	0	缓冲区中产品数量	生产时加1，消费时减1
pmtx_sem	1	生产者互斥锁	进入临界区减1，离开加1
cmtx_sem	1	消费者互斥锁	进入临界区减1，离开加1

- **prod_sem = 8**：表示初始时缓冲区有8个空位，可以放入8个产品
- **cons_sem = 0**：表示初始时缓冲区没有产品，消费者必须等待
- **互斥信号量 = 1**：表示临界区空闲，可以有一个进程进入

抽烟者问题中的信号量

信号量	初值	物理意义	值的变化
tobacco_sem	0	是否有适合烟草拥有者的材料组合	供应者提供时加1，抽烟者取用时减1
paper_sem	0	是否有适合纸拥有者的材料组合	供应者提供时加1，抽烟者取用时减1
glue_sem	0	是否有适合胶水拥有者的材料组合	供应者提供时加1，抽烟者取用时减1
supplier_sem	1	供应者是否可以工作	抽烟者完成后加1，供应者工作时减1
mutex_sem	1	桌子的互斥访问	进入临界区减1，离开加1

进程互斥和同步的实现机制

互斥的实现

互斥信号量（初值为1）确保同一时刻只有一个进程能访问共享资源：

```
down(mutex_sem);    // 进入临界区，mutex_sem: 1→0
// 临界区代码
up(mutex_sem);      // 离开临界区，mutex_sem: 0→1
```

当第二个进程试图进入临界区时：

- 执行down(mutex_sem)，mutex_sem变为-1
- 进程被阻塞，等待第一个进程释放资源
- 第一个进程执行up(mutex_sem)时，mutex_sem变为0，唤醒等待的进程

同步的实现

同步信号量协调进程间的执行顺序：

在生产者/消费者问题中：

- **prod_sem**控制生产者：只有当缓冲区不满时（prod_sem > 0）才能生产
- **cons_sem**控制消费者：只有当缓冲区不空时（cons_sem > 0）才能消费

在抽烟者问题中：

- **tobacco_sem、paper_sem、glue_sem**分别控制对应的抽烟者
- **supplier_sem**控制供应者的工作节奏

死锁的避免

通过合理的信号量操作顺序避免死锁：

正确的顺序（生产者）：

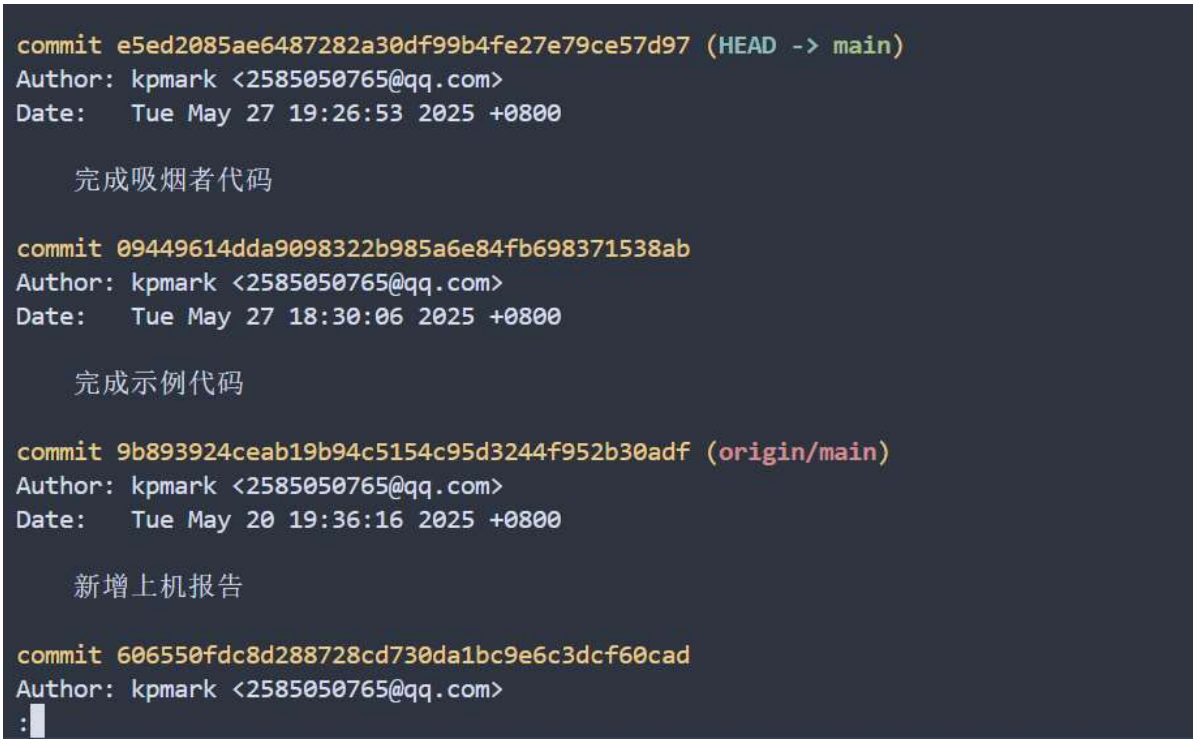
```
down(prod_sem);   // 先获取资源
down(pmtx_sem);   // 再获取互斥锁
// 临界区
up(pmtx_sem);     // 先释放互斥锁
up(cons_sem);     // 再增加资源计数
```

如果顺序错误，可能导致死锁：

- 进程A持有互斥锁等待资源
- 进程B持有资源等待互斥锁

Git 记录和上传

执行`git log`代码，得到结果如下图



markzhang12345/os-homework: 大连理工大学操作系统课程作业